Extending Knowledge-Level Contingent Planning for Robot Task Planning

Extending Knowledge-Level Contingent Planning for Robot Task Planning

Ronald P. A. Petrick
School of Informatics
University of Edinburgh
Edinburgh EH8 9AB, Scotland, UK
rpetrick@inf.ed.ac.uk

Andre Gaschler
fortiss GmbH
An-Institut Technische Universität München
Munich, Germany
gaschler@fortiss.org

Abstract

We present a set of extensions to the knowledge-level PKS (Planning with Knowledge and Sensing) planner, aimed at improving its ability to generate plans in real-world robotics domains. These extensions include a facility for integrating externally-defined reasoning processes in PKS (e.g., invoking a motion planner), an interval-based fluent representation for capturing the effects of noisy sensors and effectors, and an application programming interface (API) to facilitate software integration on robot platforms. We demonstrate our techniques in three simple robot domains, which show their applicability to a broad range of robot planning applications involving incomplete knowledge, real-world geometry, and multiple robots and sensors.

Introduction and Motivation

A robot operating in a real-world domain often needs to do so with *incomplete information* about the state of the world. A robot with the ability to *sense* the world can also gather information to generate plans with *contingencies*, allowing it to reason about the outcome of sensed data at plan time.

In this paper, we explore an application of planning with incomplete information and sensing actions to the problem of task planning in robotics domains. In particular, building models of realistic domains which can be used with general-purpose planning systems often involves working with incomplete (or uncertain) perceptual information arising from real-world sensors. Furthermore, this task may be complicated by the difficulties of bridging the gap between geometric and symbolic representations: robot systems typically reason about joint angles, spatial coordinates, and continuous spaces, while many symbolic planners work with discrete representations in represented in logic-like languages.

Our approach makes use of the PKS (Planning with Knowledge and Sensing) planner (Petrick and Bacchus 2002; 2004) as the high-level reasoning tool for task planning in robotics domains. PKS is a general-purpose contingent planner that operates at the *knowledge level* (Newell 1982), by reasoning about how its knowledge changes due to action during plan generation. PKS is able to represent known and unknown information, and model sensing actions using concise but rich domain descriptions, making it well suited for reasoning in structured, partially-known environments of the kind that arise in many robot scenarios.

Figure 1: In the FORCE SENSING scenario, a compliant robot manipulator senses if beverage containers are filled by lifting them and sensing their weight. Objects must be held upright while moving to prevent spilling, unless they are known to be completely empty or unopened.

While PKS has been used successfully in previous robot domains (Petrick et al. 2009), it lacks certain features which could improve its applicability to a wider range of robotics tasks. In this paper, we describe a set of extensions designed to improve PKS's ability to generate plans in real-world robot scenarios, by focusing on three tasks: combining high-level symbolic planning with low-level motion planning, reasoning about noisy sensors and effectors, and facilitating planner-level software integration on robot platforms.

The planner has also been integrated into a larger software framework called *Knowledge of Volumes for robot task Planning (KVP)* (Gaschler et al. 2013a), aimed at facilitating the use of planning techniques on a variety of robot platforms (see Figures 1 and 5), which has been developed as part of the JAMES project.[1] This framework serves as the basis for the robot demonstrators we describe below.

The rest of this paper is organised as follows. We first

[1] See http://james-project.eu/ for more information.

present an overview of PKS and then describe three extensions to the basic planning system which enhance its ability to operate in robotics domains. We then give three examples of robot domains where we use the extended version of the planner to generate solutions; the first two domains are tested on real robots, while the third domain is tested in simulation. Finally, we situate our approach with respect to related research and discuss future directions of our work.

Planning with Knowledge and Sensing (PKS)

PKS (Planning with Knowledge and Sensing) is a contingent planner that builds plans in the presence of incomplete information and sensing actions (Petrick and Bacchus 2002; 2004). PKS works at the *knowledge level* by reasoning about how the planner's knowledge state, rather than the world state, changes due to action. PKS works with a restricted subset of a first-order logical language, and limited inference. Thus, unlike planners that reason directly with possible worlds models or belief states, PKS works with a set of formulae representing the planner's knowledge state. This enables it to support a rich representation with features such as functions and variables; however, as a trade-off, its restricted representation means that the planner cannot model certain types of knowledge.

PKS is based on a generalisation of STRIPS (Fikes and Nilsson 1971). In STRIPS, the state of the world is modelled by a single database. Actions update this database and, by doing so, update the planner's world model. In PKS, the planner's knowledge state, rather than the world state, is represented by a set of five databases, each of which models a particular type of knowledge. The contents of these databases have a fixed, formal interpretation in a modal logic of knowledge. Actions can modify any of the databases, which has the effect of updating the planner's knowledge state. To ensure efficient inference, PKS restricts the type of knowledge (especially disjunctions) that it can represent. The contents of the databases are as follows:

$\mathbf{K_f}$: This database is like a STRIPS database except that both positive and negative facts are permitted and the closed world assumption is not applied. K_f is used for modelling action effects that change the world. K_f can include any ground literal ℓ, where $\ell \in K_f$ means "the planner knows ℓ." K_f can also contain known function (in)equality mappings.

$\mathbf{K_w}$: This database models the plan-time effects of sensing actions with binary outcomes. $\phi \in K_w$ means that at plan time the planner either "knows ϕ or knows $\neg\phi$," and that at execution time this disjunction will be resolved. In such cases we will also say that the planner "knows whether ϕ." Know-whether information is important since PKS uses such knowledge to construct conditional plans (see below).

$\mathbf{K_v}$: This database stores information about function values that will become known at execution time. In particular, K_v can model the plan-time effects of sensing actions that return constants, such as numeric values. K_v can contain any unnested function term f, where $f \in K_v$ means that at plan time the planner "knows the value of f." At execution time, the planner will have definite information about f's value. As a result, PKS is able to use K_v terms as *run-time variables* (Etzioni et al. 1992) in its plans.

$\mathbf{K_x}$: This database models the planner's *exclusive-or* knowledge. Entries in K_x have the form $(\ell_1|\ell_2|\ldots|\ell_n)$, where each ℓ_i is a ground literal. Such formulae represent a particular type of disjunctive knowledge that arises in many planning scenarios, namely that "exactly one of the ℓ_i is true."

$\mathbf{LCW}$: This database stores the planner's *local closed world* information (Etzioni, Golden, and Weld 1994), i.e., instances where the planner has complete information about the state of the world. We will not use LCW in this paper.

PKS's databases can be inspected through a set of *primitive queries* that ask simple questions about the planner's knowledge state. Simple knowledge assertions can be tested with a query $K(\phi)$ which asks: "is a formula ϕ true?" A query $K_w(\phi)$ asks whether ϕ is known to be true or known to be false (i.e., does the planner "know whether ϕ"). A query $K_v(t)$ asks "is the value of function t known?" The negation of the above queries can also be used. An inference procedure is used to evaluate primitive queries by checking the contents of the databases, taking into consideration the interaction between different types of knowledge.

An action in PKS is modelled by a set of *preconditions* that query the agent's knowledge state, and a set of *effects* that update the state. Action preconditions are simply a list of primitive queries. Action effects are described by a collection of STRIPS-style "add" and "delete" operations that modify the contents of individual databases. E.g., $\mathit{add}(K_f, \phi)$ adds ϕ to K_f, and $\mathit{del}(K_w, \phi)$ removes ϕ from K_w. Actions can also have ADL-style context-dependent effects (Pednault 1989), where the secondary preconditions of an effect are described by lists of primitive queries. A simple form of quantification, $\forall^K x$ and $\exists^K x$, that ranges over known instantiations of x can also be used. Examples of PKS actions are shown below in Figure 3.

PKS constructs plans by reasoning about actions in a simple forward-chaining manner: if the preconditions of an action are satisfied by the planner's knowledge state, then the action's effects are applied to produce a new knowledge state. Planning then continues from the resulting state. PKS can also build contingent plans with branches, by considering the possible outcomes of its K_w and K_v knowledge. For instance, if $\phi \in K_w$ then PKS can construct two conditional branches in a plan: along one branch (the K^+ branch) ϕ is assumed to be known (i.e., ϕ is added to K_f), while along the other branch (the K^- branch), $\neg\phi$ is assumed to be known (i.e., $\neg\phi$ is added to K_f). A similar type of multi-way branching plan can also be built by considering a restricted type of K_v information. Planning continues along each branch until the *goal*—a list of primitive queries–is satisfied. A sample plan with branches is shown in Figure 4, and described in greater detail below.

Extensions to PKS for Robot Task Planning

In this section we consider three recent extensions to the basic PKS system which we believe are particularly useful for robot task planning. First, we describe a mechanism which

allows externally-defined procedures (e.g., from support libraries) to be integrated with the internal reasoning mechanisms of the planner. Second, we present an extension of the PKS representation which allows a form of noisy numerical information to be modelled, for instance to represent the effects of error prone sensors. Finally, we describe a software-level application programming interface to PKS, which aids in the engineering task of integrating the planner with a robot system.

Executing Externally-Defined Procedures

The first extension we describe aims to take advantage of existing reasoning tools by providing a mechanism for PKS to invoke externally-defined procedures (e.g., defined in special purpose libraries) from within the planner's internal reasoning mechanism during plan generation. While this idea is not new, and has been successfully applied in other contexts (see the discussion section, below), the introduction of this technique into PKS is a recent extension to the planner.

In particular, PKS provides an external query mechanism of the form:

$$\texttt{extern}(\mathit{proc}(\vec{x})),$$

where `extern` is a special keyword indicating that control should be transferred to an external procedure with the name *proc*. $\vec{x}$ is a set of parameters that should be passed to *proc*. In general, x can contain any symbols defined in PKS's knowledge state, providing a link between the planner and the externally-defined procedure. An `extern` call can be used within an action definition, either as a precondition or an effect. The return value of the `extern` call, defined within the external procedure, is passed back to PKS, which interprets it in the context where it occurs in the action. Additional tests may be performed on this value, which can be assigned to domain properties and included in the planner's knowledge state. While no restrictions are placed on when such procedures can be used in a planning domain, in practice `extern` calls are most useful if used for complex or special purpose reasoning that cannot easily be modelled in the planner's restricted representation language, or where more efficient reasoning engines already exist.

The `extern` mechanism provides a powerful tool for PKS to use in robotics domains by augmenting PKS's core reasoning capabilities with the addition of motion planning, collision detection, and other special purpose robotics libraries. For instance, geometric predicates and continuous motions can be evaluated with `extern` calls, and reasoned about at the symbolic level, enabling us to solve problem instances which may be difficult to model directly at either the motion planning or symbolic planning level alone. Examples of this process are given below.

One important drawback with this facility in its present form, is that there is no control over how long an external procedure may take to execute, or whether it will terminate at all. As a result, we are currently extending our `extern` implementation to introduce a simple timeout facility that will force external procedure calls to terminate if a specified cutoff time is reached. Currently, however, the domain designer must ensure that any externally-defined procedures operate correctly in the context of a given planning domain.

Reasoning with Interval-Valued Fluents

One type of sensed information that arises in many real-world robotics contexts is *numerical* information, which is often necessary for modelling state properties (e.g., the robot is 10 metres from the wall), limited resources (e.g., ensure the robot has enough fuel), constraints (e.g., only grasp an object if its radius is less than 10 cm), or arithmetic operations (e.g., advancing the robot one step reduces its distance to the wall by 1 metre). Reasoning with incomplete numerical information is often problematic, however, especially when planners represent incompletely known state properties by sets of states, each of which denotes a possible configuration of the actual world state. E.g., if a fluent f could map to any natural number between 1 and 100, then we require 100 states to capture f's possible mappings. The state explosion resulting from large sets of mappings can be computationally difficult for planners that must reason directly with individual states to construct plans.

In PKS, we build on a previous planning approach (Petrick 2011) which uses *interval-valued fluents (IVFs)* (Funge 1998) to avoid some of the computational problems involved with uncertain numerical information. The idea is simple: instead of representing each possible mapping by a separate state, a single interval mapping is used, where the endpoints of the interval indicate the fluent's range of possible values. Thus, a fluent f that could map to values between 1 and 100 can be denoted in an interval-valued form by $f = \langle 1, 100\rangle$.

In general, PKS treats each IVF as a function whose denotation is an *interval* of the form $\langle u, v\rangle$. The *endpoints* of the interval, u and v, indicate the bounds on the range of possible mappings for the fluent. Since we are interested in planning with incomplete information, a mapping $f = \langle u, v\rangle$ will mean that the value of f is known to be in the interval $\langle u, v\rangle$. If a fluent maps to a *point interval* of the form $\langle u, u\rangle$, then the mapping is certain and known to be equal to u.

PKS's knowledge of (general) IVFs are stored in its K_x database, as a generalisation of its exclusive-or information. In addition to basic intervals, *disjunctive intervals* (i.e., sets of disjoint interval mappings) are also permitted. For instance, if a fluent f could possibly map to any value between 5 and 10 or, alternatively, map to values between 15 and 18, we can represent such information by the K_x formula $(f = \langle 5, 10\rangle \,|\, f = \langle 15, 18\rangle)$.

Certain types of IVFs can also be represented in the K_v and K_w databases. For instance, a fluent of the form $f : \langle x - c, x + c\rangle$ in K_v means that the value of the fluent f is known, and f is in the range $x \pm c$, for some numeric constant c and unknown fluent value x. This mechanism can be used to model the results of noisy sensors. In K_w, we also permit numeric relations of the form f **op** c, where **op** $\in \{=, \neq, >, <, \geq, \leq\}$ and c is a numeric constant. Thus, $f > 5 \in K_w$ can be used to model a sensing action that determines whether f is greater than 5 or not. Since K_w is used to build contingent branches into a plan, this extension also enables PKS to build branches based on IVFs.

An Application Programming Interface

The task of integrating a planner onto a robot platform often centres around the problem of representation, and how

to abstract the capabilities of a robot and its working environment so that it can be put in a suitable form for use by the planner. Integration also typically requires the ability to communicate information between system components. Thus, the integration of a planning system usually requires a consideration of certain engineering-level concerns, to ensure proper interoperability with components that aren't traditionally considered in theoretical planning settings.

In order to facilitate the task of providing software-level planning services to robot systems, we have created an application programming interface (API) for a version of PKS implemented as a C++ library. This interface abstracts many common planning operations into a series of functions which provide direct access to these services. For instance, this interface includes methods for manipulating domain representations, as well as functions for controlling certain aspects of the the plan generation process itself (e.g., selecting goals, generation strategies, or planner-specific settings). Moreover, functions that allow plans to be manipulated as first-class entities (e.g., for replanning) are provided. A fragment of the API is given in Figure 2.

Overall, the API is designed to be generic and is not meant to be tied to one particular planning system. For instance, the planner configuration methods are meant to provide a way to set certain properties of the underlying planning system, and provide access to features needed for debugging. The domain configuration functions provide the main methods for defining planning domain models, either from traditional domain/problem files, or via string-based descriptions. One important idea behind the configuration functions is that they offer the possibility of specifying domains to the planner incrementally, using function calls alone, rather than specifying a single monolithic domain file. This means that an initial domain could be specified and then later revised, for instance due to additional information discovered by the robot during execution (e.g., new domain objects, revised action descriptions, additional properties corresponding to new capabilities of the robot, etc.). Finally, the plan generation and iteration functions specify methods for controlling various aspects of the plan generation process, and provide a way for processes external to the planner to control simple monitoring and replanning activities, including updates to certain aspects of the planning problem, such as goal change.

We will discuss the integration of PKS on our robot platforms in greater detail in the discussion section below.

```
// Configuration and debugging
void    reset();
string  getPlannerProperty(string);
bool    setPlannerProperty(string, string);

// Domain configuration
bool    defineDomain(string);
bool    defineSymbols(string);
bool    defineActions(string);
bool    defineProblems(string);
bool    definePlanState(string);
bool    defineObservedState(string);

// Plan generation and iteration
bool    buildPlan();
string  getCurrentPlan();
Action  getNextAction();
bool    isNextActionEndOfPlan();
bool    isPlanDefined();
bool    setProblem(string);
bool    setProblemGoal(string);
```

Figure 2: A fragment of the PKS API.

Example Domains

To demonstrate our approach, we now describe three robotics scenarios that make use of knowledge-level planning: the FORCE SENSING and the BIMANUAL robot scenarios, based on domains first described in (Gaschler et al. 2013c) and tested on real robots, and the ROBOT LOCALISATION scenario, tested in simulation. In all scenarios, the robot uses sensing actions to obtain knowledge of some domain property which is necessary for achieving the goal. In the first scenario, only the basic PKS system is used. In the second scenario, PKS's external procedure mechanism is used to link a motion planning library to the planner's internal reasoning mechanisms. In the final scenario, we make use of interval-valued fluents in a simple localisation task. In each case, we discuss the symbolic domain definitions of the scenario, and provide an example of the solution plan that was generated in that domain.

Force Sensing Scenario

In the FORCE SENSING scenario, a robot manipulator is tasked with transferring beverage containers from one table to another, as shown in Figure 1. Through its torque sensors, it can sense the external force of a grasped container, and decide whether or not that drink could be spilled. The robot should hold drinks exactly upright to prevent spilling, unless a drink is known to be completely empty, in which case a faster arbitrary motion may be performed. In order to keep this scenario simple, the location of all objects are known and no sensing except force sensing is available.

Figure 3 shows the PKS actions in the FORCE SENSING scenario, which includes a sensing action, `senseWeight`, which senses the weight of a beverage container `?o`. To perform this action, the robot must first be grasping object `?o`. To ensure only new knowledge is gained from this action, and to increase planning efficiency, we include a precondition that the robot must not yet know whether `?o` is spillable. When this action is performed, knowledge of whether `?o` is spillable or not is added to PKS's K_w database.

This scenario also includes a number of actions for manipulating domain objects, including `transferUpright`, `transfer`, `grasp`, and `ungrasp` actions, also listed in Figure 3. For example, in the `transferUpright` action, the robot can move a grasped container from one table to the other, while keeping the orientation of its parallel gripper fixed. Only objects that are grasped and not yet removed to the second table can be transferred.

An example plan for the FORCE SENSING scenario is shown in Figure 4 for the case of two objects in the domain. In particular, a sensing action is performed on each

```
action senseWeight(?o:object)
    preconds:
        ¬K_w(isSpillable(?o)) &
        K(isGrasped(?o))
    effects:
        add(K_w, isSpillable(?o))

action transfer(?o:object)
    preconds:
        K(¬isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action transferUpright(?o:object)
    preconds:
        K(isSpillable(?o)) &
        K(isGrasped(?o)) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isRemoved(?o))

action grasp(?o:object)
    preconds:
        K(emptyGripper) &
        K(¬isRemoved(?o))
    effects:
        add(K_f, isGrasped(?o)),
        add(K_f, ¬emptyGripper)

action ungrasp(?o:object)
    preconds:
        K(isGrasped(?o)) &
        K(isRemoved(?o))
    effects:
        add(K_f, ¬isGrasped(?o)),
        add(K_f, emptyGripper)
```

Figure 3: Actions in the FORCE SENSING domain.

object (`can1` and `can2`) and the objects are individually manipulated depending on whether their contents are spillable or not. The resulting plan therefore considers four contingent situations which could arise during plan execution. This scenario was tested on a joint-impedance controlled lightweight 7-DoF robot with a force-controlled parallel gripper. Forces were measured by internal torque sensing.

Bimanual Robot Scenario

The second scenario is a demonstration of a BIMANUAL robot (Figure 5) whose hands can reach different areas of a table. In this case, the robot can sense if bottles on the table are empty or full using a top-down camera. Its goal is to clean up all empty bottles by removing them to a certain "dishwasher" location. In order to achieve this goal, the robot must move objects that are only accessible by its left arm to a location that its right arm can reach, a behaviour which arises purely from symbolic planning. In contrast to the previous FORCE SENSING scenario, the BIMANUAL robot scenario relies on visual information, which can be gathered without requiring manipulation.

```
 1.  grasp(can1) ;
 2.  senseWeight(can1) ;
 3.  branch(isSpillable(can1))
 4.  K+:
 5.      transferUpright(can1) ;
 6.      ungrasp(can1) ;
 7.      grasp(can2) ;
 8.      senseWeight(can2) ;
 9.      branch(isSpillable(can2))
10.      K+:
11.          transferUpright(can2) ;
12.          ungrasp(can2).
13.      K-:
14.          transfer(can2) ;
15.          ungrasp(can2).
16.  K-:
17.      transfer(can1) ;
18.      ungrasp(can1) ;
19.      grasp(can2) ;
20.      senseWeight(can2) ;
21.      branch(isSpillable(can2))
22.      K+:
23.          transferUpright(can2) ;
24.          ungrasp(can2).
25.      K-:
26.          transfer(can2) ;
27.          ungrasp(can2).
```

Figure 4: A plan for removing 2 objects from a table in the FORCE SENSING domain.

The PKS actions in the BIMANUAL scenario are given in Figure 6. Two robot arms are tasked with removing all empty bottles that are visible on a table, and moving them to the dishwasher location, which can only be reached by the right robot arm. The domain includes one sensing action, `senseIfEmpty`, which has no precondition other than the requirement that the knowledge it gathers must be new. For manipulation, both robot arms can perform the `pickUp` and `putDown` actions. However, not all locations can be reached by both hands, so the preconditions of these actions include an **extern** call to `isReachable`, which is defined in a motion planning library and which checks reachability for a specific manipulator and location. This interaction of symbolic and motion planners is described in greater detail in the discussion section below, and in (Gaschler et al. 2013a).

An example plan is shown in Figure 7 for the case of 4 objects. In particular, the plan senses each object to detect whether or not it is empty and then constructs a conditional plan to subsequently remove the empty objects to the dishwasher. The resulting plan therefore considers 16 possible configurations of empty/non-empty bottles which could arise at execution. (The actions for the case where `bottle0` and `bottle2` are empty are shown.) It is interesting to observe that this simple robot scenario already gives rise to interesting behaviour: since the right arm cannot directly reach all objects that need to be transferred to the goal location, the left arm must pass those objects to a location reachable by both hands. This behaviour has not been pre-programmed, but instead arises purely from the combination of symbolic and geometric planning.

Figure 5: In the BIMANUAL scenario, a camera is used to recognise empty bottles which a bimanual robot should remove from the table to a "dishwasher" location on the left side, behind the table (Gaschler et al. 2013a; Giuliani et al. 2013). A video of the robot operating in this scenario is available at `http://youtu.be/yMmZkhHr8ss`.

This domain was tested on a two 6-DoF industrial manipulator setup with Meka Robotics H2 humanoid hands, with an RGB camera facing top-down for simple colour-filtering object recognition, as described in (Foster et al. 2012).

Robot Localisation Scenario

In the final example, we consider a robot whose location, represented by the IVF `robotLoc`, is measured by the robot's distance to a wall. The robot has two physical actions available to it: `moveForward`, which moves the robot either 1 or 2 units towards the wall; and `moveBackward`, which moves the robot 1 unit away from the wall. The robot also has a sensing action, `atTarget`, which senses whether the robot is at a target location, specified by the function `targetLoc`. Additionally, the robot also has a second sensing action, `withinTarget`, that determines whether or not the robot is within the target distance `targetLoc`.

The definitions of the PKS actions for this scenario are given in Figure 8 (all action preconditions are assumed to be true). The robot's initial location is specified by the interval mapping $\texttt{robotLoc} = \langle 3, 4\rangle$ stored in K_x. The goal is to move the robot to the target location, i.e., $K(\texttt{robotLoc} = \texttt{targetLoc})$, where $\texttt{targetLoc} = 2$ is stored in K_f.

One solution generated by PKS is the conditional plan in Figure 9. Since forward movements may change the robot's position by either 1 unit or 2 units, `noisyForward` in step 1 results in an even less certain position for the robot, namely $\texttt{robotLoc} = \langle 1, 3\rangle \in K_x$. However, the sensing action in step 2, together with the branch point in step 3, lets us split this interval into two parts. In step 4, we assume that $\texttt{robotLoc} \leq 2$ and consider the case where $\texttt{robotLoc} = \langle 1, 2\rangle$. `atTarget`, together with the branch in step 6, lets us divide this interval even further: in step 7, $\texttt{robotLoc} = 2$ and the goal is satisfied, while in step 8, $\texttt{robotLoc} = 1$ and a `moveBackward` action achieves the goal. In step 10 we consider the other sub-interval of the first branch, i.e., $\texttt{robotLoc} = 3 \in K_f$. In this case we have definite knowledge, however, a subsequent `noisyForward` results in $\texttt{robotLoc} = \langle 1, 2\rangle$. The remainder of the plan in steps 12–16 is the same as in steps 5–9: the robot conditionally moves backwards in the case that `robotLoc` is determined to be 1, while the plan trivially achieves the goal if $\texttt{robotLoc} = 2$.

We have not tested this domain on a real robot yet but have instead performed a series of tests in simulation using a variety of initial and target locations. Experimentation with IVF domains on a real robot is a focus of current work.

```
action senseIfEmpty(?o:object)
    preconds:
        ¬K_w(isEmptyBottle(?o))
    effects:
        add(K_w, isEmptyBottle(?o))

action pickUp(?r:robot, ?o:object, ?l:location)
    preconds:
        K(?l = getObjectLocation(?o)) &
        K(handEmpty(?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, ?l = getObjectLocation(?o)),
        del(K_f, handEmpty(?r)),
        add(K_f, inHand(?o, ?r))

action putDown(?r:robot, ?o:object, ?l:location)
    preconds:
        K(inHand(?o, ?r)) &
        K(extern(isReachable(?l, ?r)))
    effects:
        del(K_f, inHand(?o, ?r)),
        add(K_f, ?l = getObjectLocation(?o)),
        add(K_f, handEmpty(?r))
```

Figure 6: Actions in the BIMANUAL domain.

Related Work and Discussion

Applications of automated planning to robotics go back to the early 1980s, for instance with the famous robot systems Shakey (Nilsson 1984) and Handey (Lozano-Pérez et al. 1989). Since that time, the field has made substantial progress, and various approaches to robot task planning have been proposed, including probabilistic techniques from artificial intelligence (Kaelbling and Lozano-Pérez 2013), closed-world symbolic planning (Cambon, Alami, and Gravot 2009; Plaku and Hager 2010; Dornhege et al. 2009b), formal synthesis (Kress-Gazit and Pappas 2008; Cheng et al. 2012), and sampling-based manipulation planning (Zacharias, Borst, and Hirzinger 2006; Barry 2013).

As part of our work to apply general-purpose planning in robotics domains, we developed the *Knowledge of Volumes framework for robot task Planning (KVP)*, initially presented in (Gaschler et al. 2013a). KVP uses PKS as its underlying symbolic planner, and combines it with the idea of treating 3D geometric volumes as an intermediary representation between continuously-valued robot motions and discrete symbolic actions, to address the problem of bridg-

```
 1.  senseIfEmpty(bottle0) ;
 2.  senseIfEmpty(bottle1) ;
 3.  senseIfEmpty(bottle2) ;
 4.  senseIfEmpty(bottle3) ;
 5.  branch(isEmptyBottle(bottle0))
 6.  K+:
 7.    branch(isEmptyBottle(bottle1))
 8.    K+: ...
 9.    K-:
10.      branch(isEmptyBottle(bottle2))
11.      K+:
12.        branch(isEmptyBottle(bottle3))
13.        K+: ...
14.        K-:
15.            pickUp(left,bottle0,l0) ;
16.            putDown(left,bottle0,l5) ;
17.            pickUp(right,bottle2,l2) ;
18.            putDown(right,bottle2,dishwasher) ;
19.            pickUp(right,bottle0,l5) ;
20.            putDown(right,bottle0,dishwasher).
21.      K-: ...
22.  K-: ...
```

Figure 7: A plan for 4 objects in the BIMANUAL domain.

```
action moveForward
    effects:
        add(K_f, robotLoc := robotLoc - <1,2>)

action moveBackward
    effects:
        add(K_f, robotLoc := robotLoc + 1)

action atTarget
    effects:
        add(K_w, robotLoc = targetLoc)

action withinTarget
    effects:
        add(K_w, robotLoc <= targetLoc)
```

Figure 8: Actions in the LOCALISATION domain.

ing the gap between geometric and symbolic planning representations. By using the intermediate representation of volumes, KVP can model continuous geometry, in contrast to arbitrary discretisation (Gaschler et al. 2013a).

Previous work described the KVP framework (Gaschler et al. 2013a), and gave details of the swept volume computation for convex sets of polyhedra (Gaschler et al. 2013b). The two task planning scenarios discussed in this paper were previously presented in (Gaschler et al. 2013c), however, the present paper focuses on the planning aspects of this work, giving a detailed discussion of knowledge-level planning, sensing actions, and discrete uncertainty.

A number of approaches also address the problem of integrating symbolic planning and motion planning. For instance, our work is in part inspired by Kaelbling and Lozano-Pérez's earlier work on hierarchical task and motion planning (Kaelbling and Lozano-Pérez 2011), borrowing the continuous geometry of swept volumes. However, while the

```
                                          robotLoc
 0.                                       ⟨3, 4⟩
 1.  noisyForward ;                       ⟨1, 3⟩
 2.  withinTarget ;
 3.  branch(robotLoc ≤ targetLoc)
 4.  K+:                                  ⟨1, 2⟩
 5.      atTarget ;
 6.      branch(robotLoc = targetLoc)
 7.      K+: nop.                           2
 8.      K-:                                1
 9.          moveBackward.                  2
10.  K-:                                    3
11.      noisyForward ;                     ⟨1, 2⟩
12.      atTarget ;
13.      branch(robotLoc = targetLoc)
14.      K+: nop.                           2
15.      K-:                                1
16.          moveBackward.                  2
```

Figure 9: A plan in the LOCALISATION domain.

geometric preconditions may be similar, their underlying aggressively hierarchical planning strategy differs from the knowledge-level planning approach we use here. Further approaches that integrate symbolic and geometric reasoning are presented by Cambon, Alami and Gravot (2009), handling geometric preconditions and effects; Dornhege et al. (2009b); and, more recently, Plaku and Hager (2010), which additionally allow differential motion constraints in a sampling-based motion and action planner. We note that the latter three approaches assume a closed world, where all symbols must be either true or false. Our approach instead represents open-world knowledge, which allows us to model incomplete information and high-level sensing. Prior work has also used PKS to connect robot vision and grasping with automated planning (Petrick et al. 2009).

In terms of our extensions to PKS, the ability to link external libraries to internal reasoning processes is key to our approach. While this idea is not new, and has been previously applied (Eiter et al. 2006; Dornhege et al. 2009a; Erdem et al. 2011), the introduction of such techniques to PKS is a recent addition to the planner. Current work is focused on extending this interface, to allow external procedures partial access to internal PKS planning states, for more efficient external execution during plan generation.

Interval-valued numeric models have been previously investigated in planning contexts, e.g., for modelling time as a resource (Edelkamp 2002; Frank and Jónsson 2003; Laborie 2003). A similar representation to ours for bounding noisy numeric properties has also been proposed by Poggioni, Milani, and Baioletti (2003). This idea also has parallels to work on register models (van Eijck 2013). The importance of numerical reasoning in planning has been recognised with the inclusion of numeric state variables in PDDL, and in planners like MetricFF. We believe representations such as our IVF approach offer a useful middle ground between discrete and fully probabilistic models of uncertainty.

Motion planning and collision detection in our work rely heavily on the Robotics Library (RL)[2] (Rickert 2011), ex-

[2]Available from `http://www.roboticslibrary.org/`.

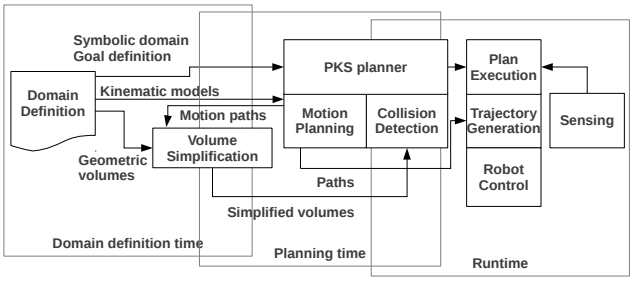

Figure 10: Overview of the implemented KVP framework (Gaschler et al. 2013a).

tended with several crucial additions to swept volume computations with sets of convex bodies (Gaschler et al. 2013a). To efficiently generate these sets of convex bodies, Mamou and Ghorbel's approximate convex decomposition algorithm (Mamou and Ghorbel 2009) is applied. An overview of KVP's component architecture is shown in Figure 10; the integration of PKS within this framework is achieved using the API described in this paper.

Finally, we note that the set of functions we defined for our planning API can be thought of as an interface to a series of abstract planning services which are ultimately implemented by some underlying "black box" planning system. As with other types of complex software modules, such an interface removes the need for the application programmer to know about how such services are actually implemented within the black box, but instead allows the designer to build more complex components that simply make use of these services. We are currently exploring the option of adapting other existing planners to use our interface, in order to experiment with alternative planner backends.

Conclusions

We described a set of extensions to PKS, aimed at improving its applicability to problems in robot task planning. We demonstrated the capabilities of our approach in solving typical robot tasks at the knowledge level, including the combination of high-level symbolic planning with low-level motion planning. Our evaluation included two simple scenarios that covered force sensing and visual sensing, with real execution on physical robot setups. A final example demonstrated a simple robot localisation task in simulation. As part of our ongoing and future work, we are continuing to refine our extensions and apply them in more complex scenarios, in order to gather empirical data and better understand the limits of our techniques. Overall, we believe our approach is useful for a broad range of robot planning applications that require incomplete knowledge, real-world geometry, and multiple robots and sensors.

Acknowledgements

This research was supported in part by the European Commission's Seventh Framework Programme under grant no. 270435 (JAMES, `james-project.eu`) and grant no. 270273 (XPERIENCE, `xperience.org`)

References

Barry, J. L. 2013. *Manipulation with Diverse Actions*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Cambon, S.; Alami, R.; and Gravot, F. 2009. A hybrid approach to intricate motion, manipulation and task planning. *International Journal of Robotics Research* 28(1):104–126.

Cheng, C.; Geisinger, M.; Ruess, H.; Buckl, C.; and Knoll, A. 2012. Game solving for industrial automation and control. In *IEEE Int. Conf. on Robotics and Automation*, 4367–4372.

Dornhege, C.; Eyerich, P.; Keller, T.; Trüg, S.; Brenner, M.; and Nebel, B. 2009a. Semantic attachments for domain-independent planning systems. In *Proc. of the Int. Conference on Automated Planning and Scheduling (ICAPS)*, 114–121.

Dornhege, C.; Gissler, M.; Teschner, M.; and Nebel, B. 2009b. Integrating symbolic and geometric planning for mobile manipulation. In *IEEE International Workshop on Safety, Security & Rescue Robotics*, 1–6.

Edelkamp, S. 2002. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research* 20:195–238.

Eiter, T.; Ianni, G.; Schindlauer, R.; and Tompits, H. 2006. Effective integration of declarative rules with external evaluations for semantic-web reasoning. In *The Semantic Web: Research and Applications*, 273–287.

Erdem, E.; Haspalamutgil, K.; Palaz, C.; Patoglu, V.; and Uras, T. 2011. Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation. In *Int. Conference on Robotics and Automation*, 4575–4581.

Etzioni, O.; Hanks, S.; Weld, D.; Draper, D.; Lesh, N.; and Williamson, M. 1992. An approach to planning with incomplete information. In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, 115–125.

Etzioni, O.; Golden, K.; and Weld, D. 1994. Tractable closed world reasoning with updates. In *Proc. of the International Conference on Knowledge Representation and Reasoning (KR)*, 178–189.

Fikes, R., and Nilsson, N. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Foster, M. E.; Gaschler, A.; Giuliani, M.; Isard, A.; Pateraki, M.; and Petrick, R. 2012. Two people walk into a bar: Dynamic multi-party social interaction with a robot agent. In *Proceedings of the ACM International Conference on Multimodal Interaction (ICMI)*.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Journal of Constraints, Special Issue on Constraints and Planning* 8:339–364.

Funge, J. 1998. Interval-valued epistemic fluents. In *AAAI Fall Symposium on Cognitive Robotics*, 23–25.

Gaschler, A.; Petrick, R.; Giuliani, M.; Rickert, M.; and Knoll, A. 2013a. KVP: A Knowledge of Volumes Approach to Robot Task Planning. In *IEEE/RSJ Intl Conf on Intelligent Robots and Systems (IROS)*, 202–208.

Gaschler, A.; Petrick, R.; Kröger, T.; Khatib, O.; and Knoll, A. 2013b. Robot task and motion planning with sets of convex polyhedra. In *RSS Workshop on Combined Robot Motion Planning and AI Planning for Practical Applications*.

Gaschler, A.; Petrick, R.; Kröger, T.; Knoll, A.; and Khatib, O. 2013c. Robot task planning with contingencies for run-time sensing. In *ICRA Workshop on Combining Task and Motion Planning*.

Giuliani, M.; Petrick, R.; Foster, M. E.; Gaschler, A.; Isard, A.; Pateraki, M.; and Sigalas, M. 2013. Comparing task-based and

socially intelligent behaviour in a robot bartender. In *Proceedings of the International Conference on Multimodal Interaction (ICMI)*.

Kaelbling, L. P., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation (ICRA)*, 1470–1477.

Kaelbling, L. P., and Lozano-Pérez, T. 2013. Integrated task and motion planning in belief space. *International Journal of Robotics Research* 32(9–10):1194–1227.

Kress-Gazit, H., and Pappas, G. 2008. Automatically synthesizing a planning and control subsystem for the darpa urban challenge. In *Automation Science and Engineering, 2008. CASE 2008. IEEE International Conference on*, 766–771.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence* 143:151–188.

Lozano-Pérez, T.; Jones, J.; Mazer, E.; and O'Donnell, P. 1989. Task-level planning of pick-and-place robot motions. *Computer* 22(3):21–29.

Mamou, K., and Ghorbel, F. 2009. A simple and efficient approach for 3d mesh approximate convex decomposition. In *IEEE International Conference on Image Processing (ICIP)*, 3501–3504.

Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18(1):87–127.

Nilsson, N. 1984. Shakey the robot. Technical Report 323, AI Center, SRI International.

Pednault, E. P. D. 1989. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning (KR)*, 324–332.

Petrick, R., and Bacchus, F. 2002. A knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Artificial Intelligence Planning and Scheduling (AIPS)*, 212–221.

Petrick, R., and Bacchus, F. 2004. Extending the knowledge-based approach to planning with incomplete information and sensing. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2–11.

Petrick, R.; Kraft, D.; Krüger, N.; and Steedman, M. 2009. Combining cognitive vision, knowledge-level planning with sensing, and execution monitoring for effective robot control. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, 58–65.

Petrick, R. 2011. An extension of knowledge-level planning to interval-valued functions. In *AAAI 2011 Workshop on Generalized Planning*.

Plaku, E., and Hager, G. 2010. Sampling-based motion planning with symbolic, geometric, and differential constraints. In *IEEE Int. Conference on Robotics and Automation*, 5002–5008.

Poggioni, V.; Milani, A.; and Baioletti, M. 2003. Managing interval resources in automated planning. *Journal of Information Theories and Applications* 10:211–218.

Rickert, M. 2011. *Efficient Motion Planning for Intuitive Task Execution in Modular Manipulation Systems*. Dissertation, Technische Universität München.

van Eijck, J. 2013. Elements of epistemic crypto logic. Slides from a talk at the LogiCIC Workshop, Amsterdam.

Zacharias, F.; Borst, C.; and Hirzinger, G. 2006. Bridging the gap between task planning and path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 4490–4495.